Assignment -4 Report submitted for
Artificial Intelligence (UNC504) by

**Name of student: Prabhnoor Singh**
**Roll number: 102115059**
**Group: 3NC3**

**Submittedto**

**Ms.Tanvi Dovedi**

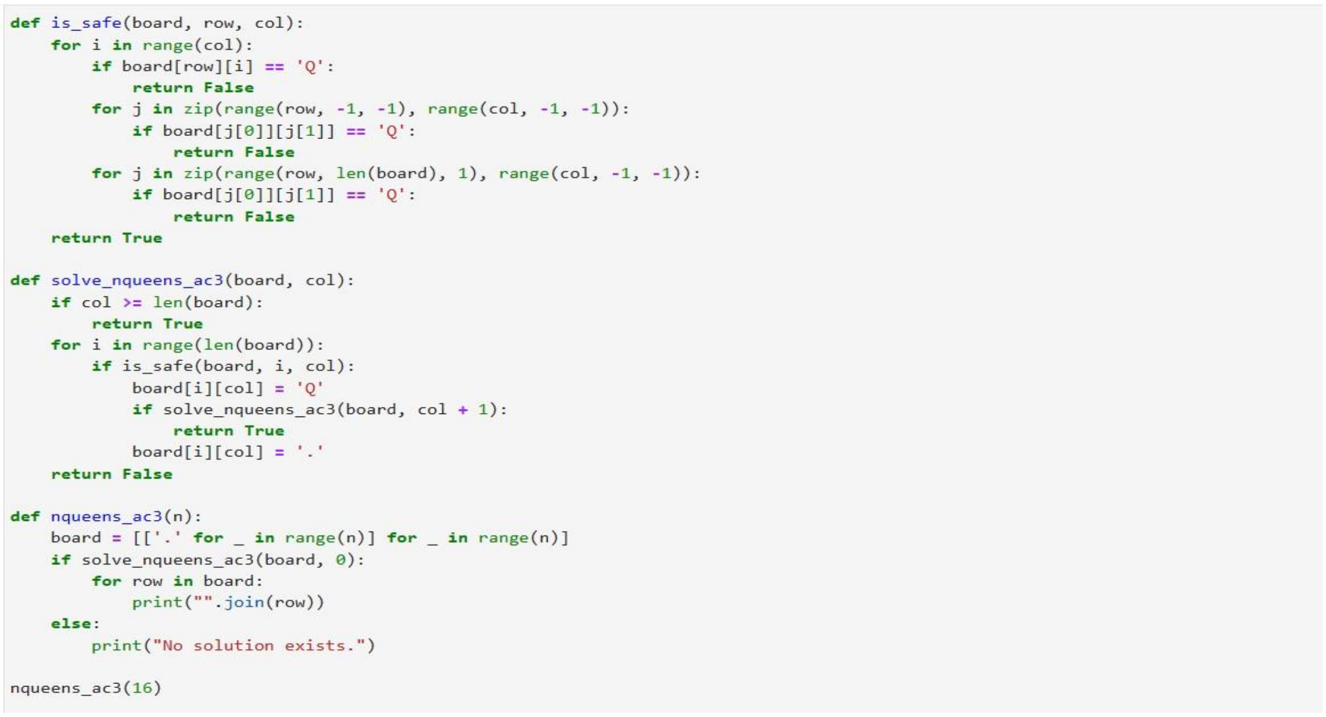
THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

**DEPARTMENTOFELECTRONICSANDCOMMUNICATIONENGINEERING**
**THAPARINSTITUTEOFENGINEERINGANDTECHNOLOGY,(ADEEMEDTOBE UNIVERSITY), PATIALA, PUNJAB**
**INDIA**
**July-Dec 2023**

# Assignment 4:

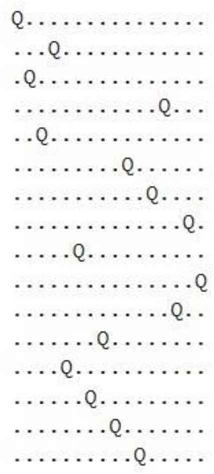
Write Python code to implement the following algorithms for solving n-queen problem

- AC-3 Algorithm

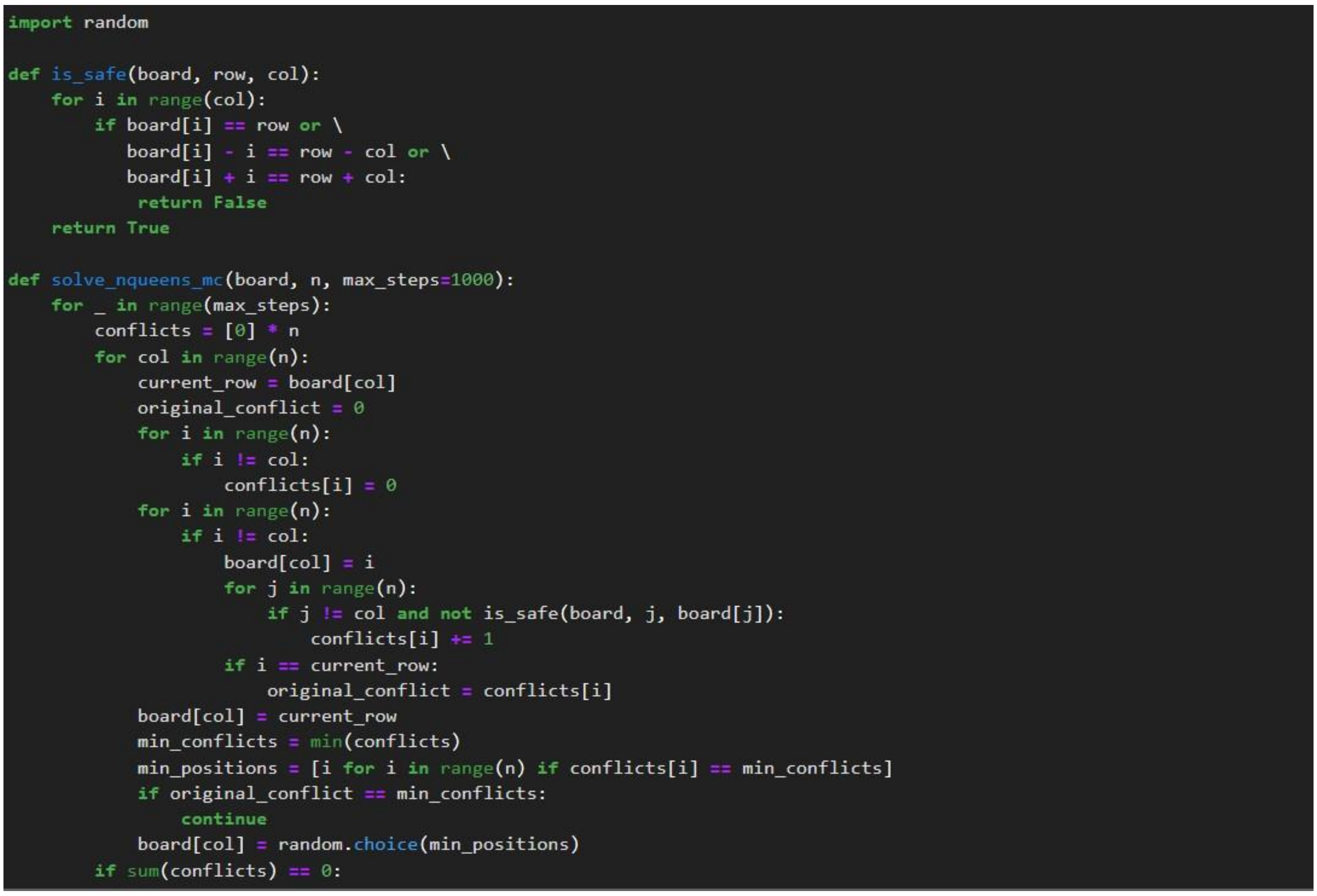


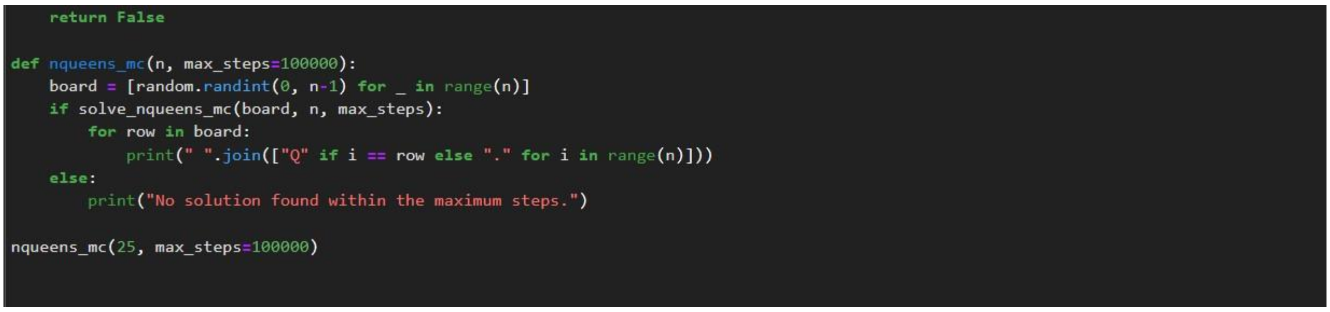
```python
def is_safe(board, row, col):
    for i in range(col):
        if board[row][i] == 'Q':
            return False
        for j in zip(range(row, -1, -1), range(col, -1, -1)):
            if board[j[0]][j[1]] == 'Q':
                return False
        for j in zip(range(row, len(board), 1), range(col, -1, -1)):
            if board[j[0]][j[1]] == 'Q':
                return False
    return True

def solve_nqueens_ac3(board, col):
    if col >= len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 'Q'
            if solve_nqueens_ac3(board, col + 1):
                return True
            board[i][col] = '.'
    return False

def nqueens_ac3(n):
    board = [['.' for _ in range(n)] for _ in range(n)]
    if solve_nqueens_ac3(board, 0):
        for row in board:
            print("".join(row))
    else:
        print("No solution exists.")

nqueens_ac3(16)
```

Output:


```
Q...............
...Q............
.Q..............
............Q...
..Q.............
.........Q......
...........Q....
..............Q.
.....Q..........
...............Q
.............Q..
.......Q........
....Q...........
......Q.........
........Q.......
..........Q.....
```

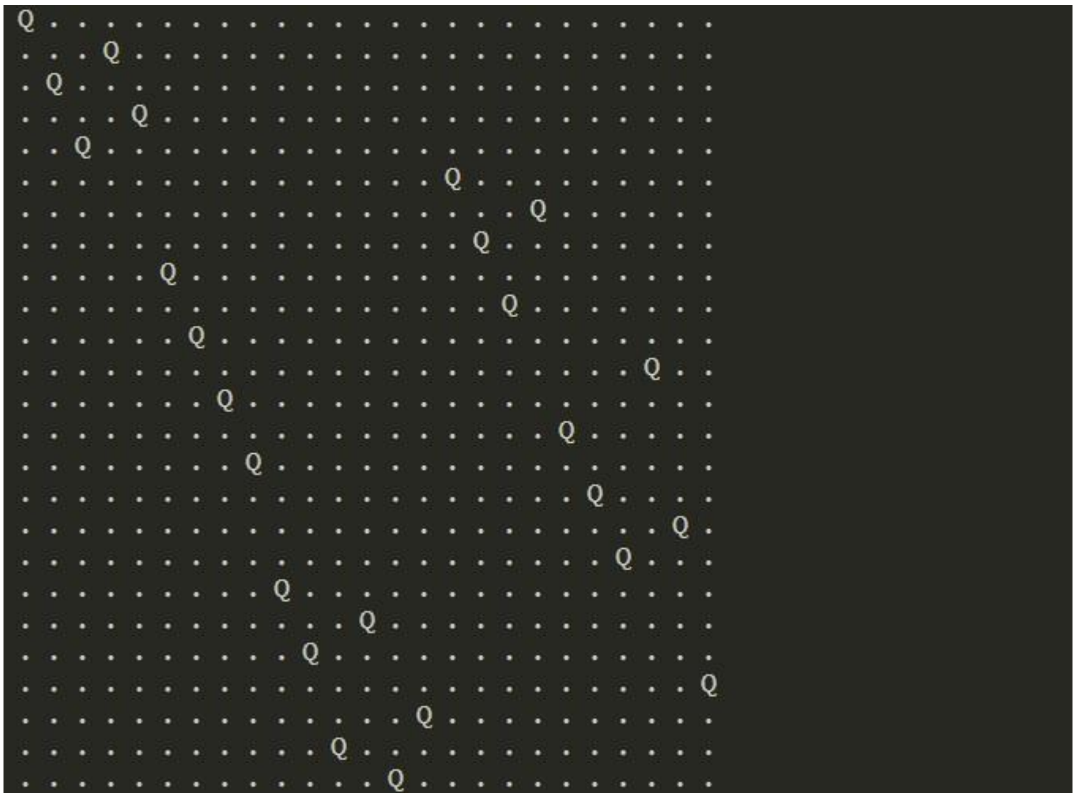
- Minimum conflict algorithm.


```python
import random

def is_safe(board, row, col):
    for i in range(col):
        if board[i] == row or \
           board[i] - i == row - col or \
           board[i] + i == row + col:
            return False
    return True

def solve_nqueens_mc(board, n, max_steps=1000):
    for _ in range(max_steps):
        conflicts = [0] * n
        for col in range(n):
            current_row = board[col]
            original_conflict = 0
            for i in range(n):
                if i != col:
                    conflicts[i] = 0
            for i in range(n):
                if i != col:
                    board[col] = i
                    for j in range(n):
                        if j != col and not is_safe(board, j, board[j]):
                            conflicts[i] += 1
                    if i == current_row:
                        original_conflict = conflicts[i]
            board[col] = current_row
            min_conflicts = min(conflicts)
            min_positions = [i for i in range(n) if conflicts[i] == min_conflicts]
            if original_conflict == min_conflicts:
                continue
            board[col] = random.choice(min_positions)
        if sum(conflicts) == 0:
```



```python
    return False

def nqueens_mc(n, max_steps=100000):
    board = [random.randint(0, n-1) for _ in range(n)]
    if solve_nqueens_mc(board, n, max_steps):
        for row in board:
            print(" ".join(["Q" if i == row else "." for i in range(n)]))
    else:
        print("No solution found within the maximum steps.")

nqueens_mc(25, max_steps=100000)
```

Output:



```
Q . . . . . . . . . . . . . . . . . . . . . . . .
. . . Q . . . . . . . . . . . . . . . . . . . . .
. Q . . . . . . . . . . . . . . . . . . . . . . .
. . . . Q . . . . . . . . . . . . . . . . . . . .
. . Q . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . Q . . . . . . . . .
. . . . . . . . . . . . . . . . . . Q . . . . . .
. . . . . . . . . . . . . . . . Q . . . . . . . .
. . . . . Q . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . Q . . . . . . .
. . . . . . Q . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . Q . .
. . . . . . . Q . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . Q . . . . .
. . . . . . . . Q . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . Q . . . .
. . . . . . . . . . . . . . . . . . . . . . . Q .
. . . . . . . . . . . . . . . . . . . . . Q . . .
. . . . . . . . . Q . . . . . . . . . . . . . . .
. . . . . . . . . . . . Q . . . . . . . . . . . .
. . . . . . . . . . Q . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . Q
. . . . . . . . . . . . . . Q . . . . . . . . . .
. . . . . . . . . . . Q . . . . . . . . . . . . .
. . . . . . . . . . . . . Q . . . . . . . . . . .
```